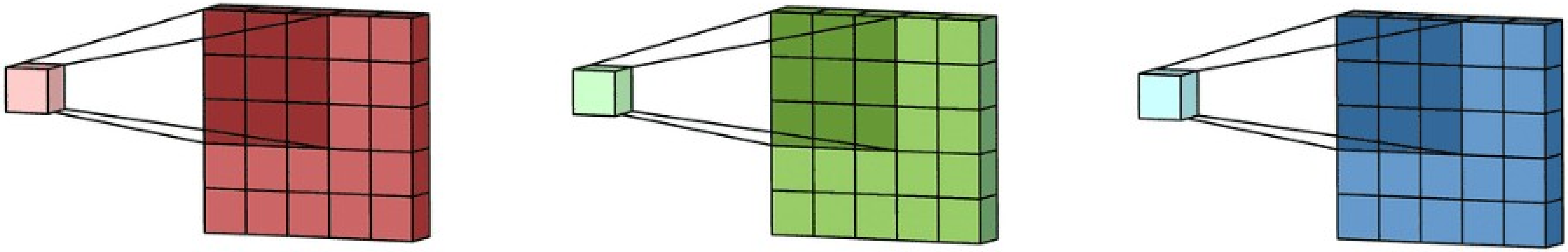
1x1 convolutions

Normal convolutions



https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

1x1 convolution

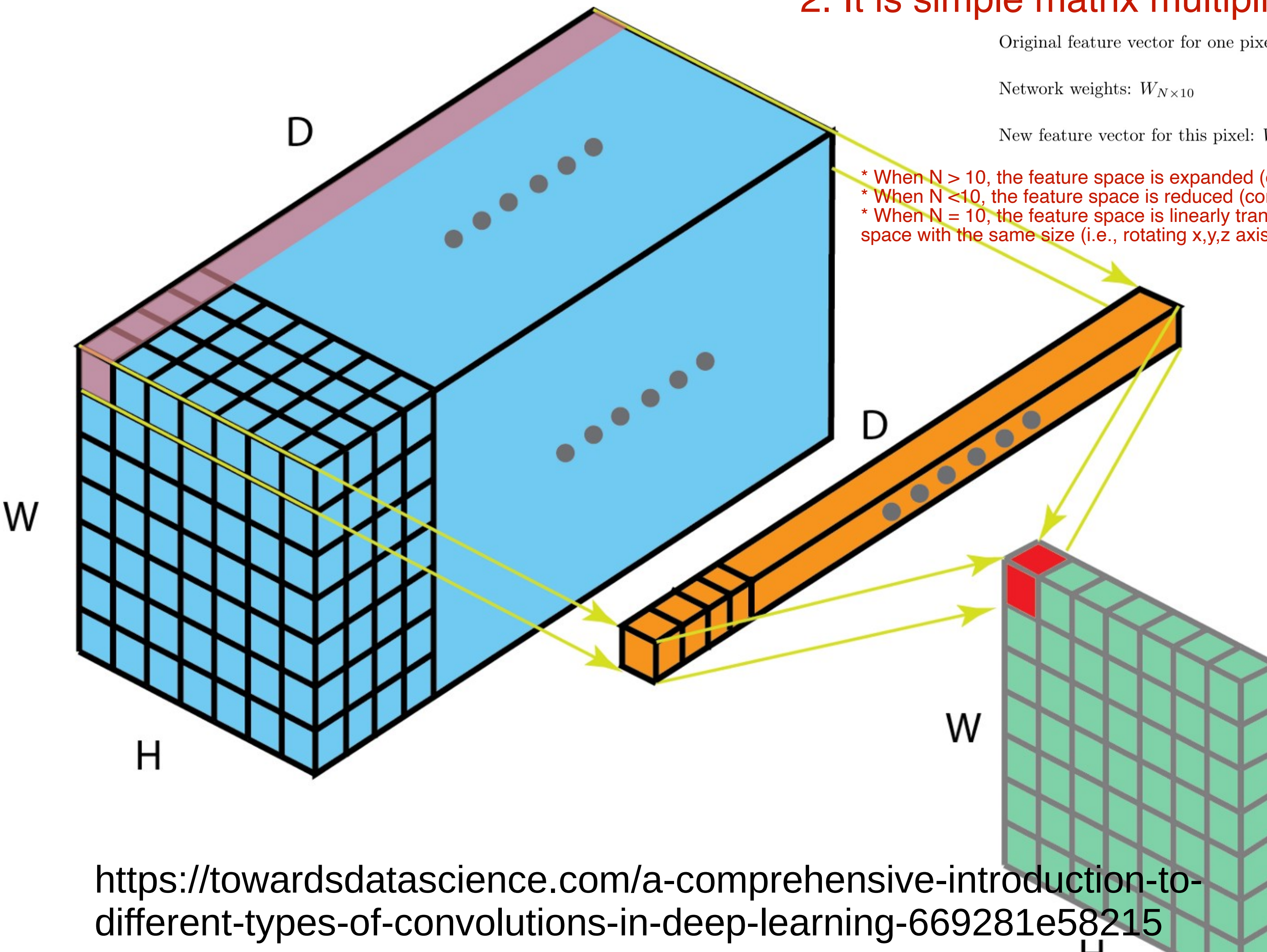
1. Think about PCA
2. It is simple matrix multiplication:

Original feature vector for one pixel: $\vec{x}_{10\times1}$

Network weights: $W_{N\times10}$

New feature vector for this pixel: $W_{N\times10} * \vec{x}_{10\times1} = \vec{x}^{new}_{N\times1}$

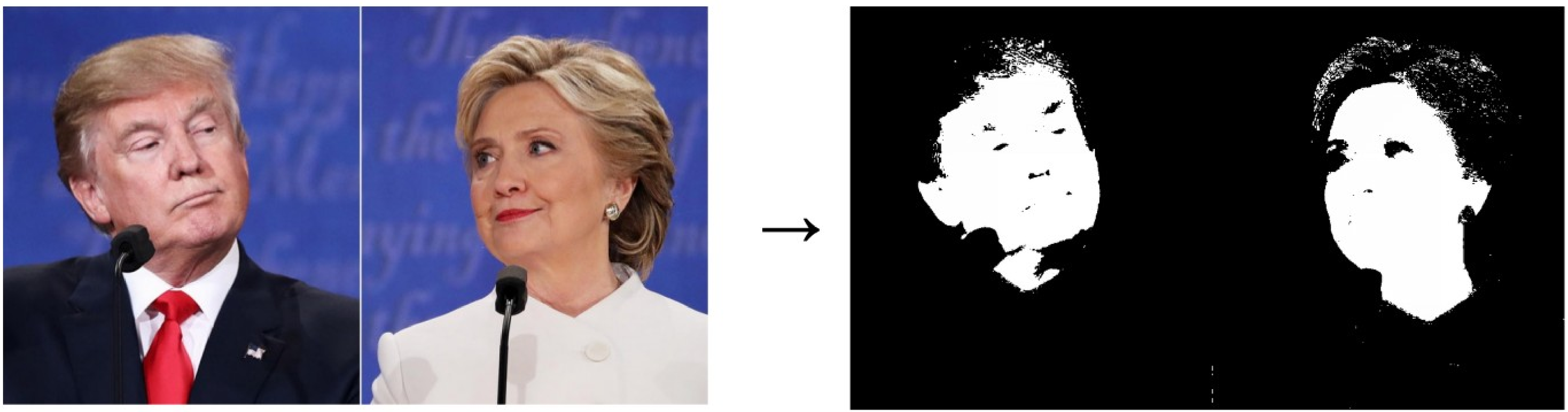
* When N > 10, the feature space is expanded (enriching)
* When N <10, the feature space is reduced (compression)
* When N = 10, the feature space is linearly transformed to a new feature space with the same size (i.e., rotating x,y,z axis)



https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Homework

Data per-processing



Colorspaces

RGB → red, green, and blue light are added together in various ways to reproduce a broad array of colors

LAB:

Defined by the International Commission on Illumination (CIE) in 1976

“predicts which spectral power distributions will be perceived as the same color”

Color spaces

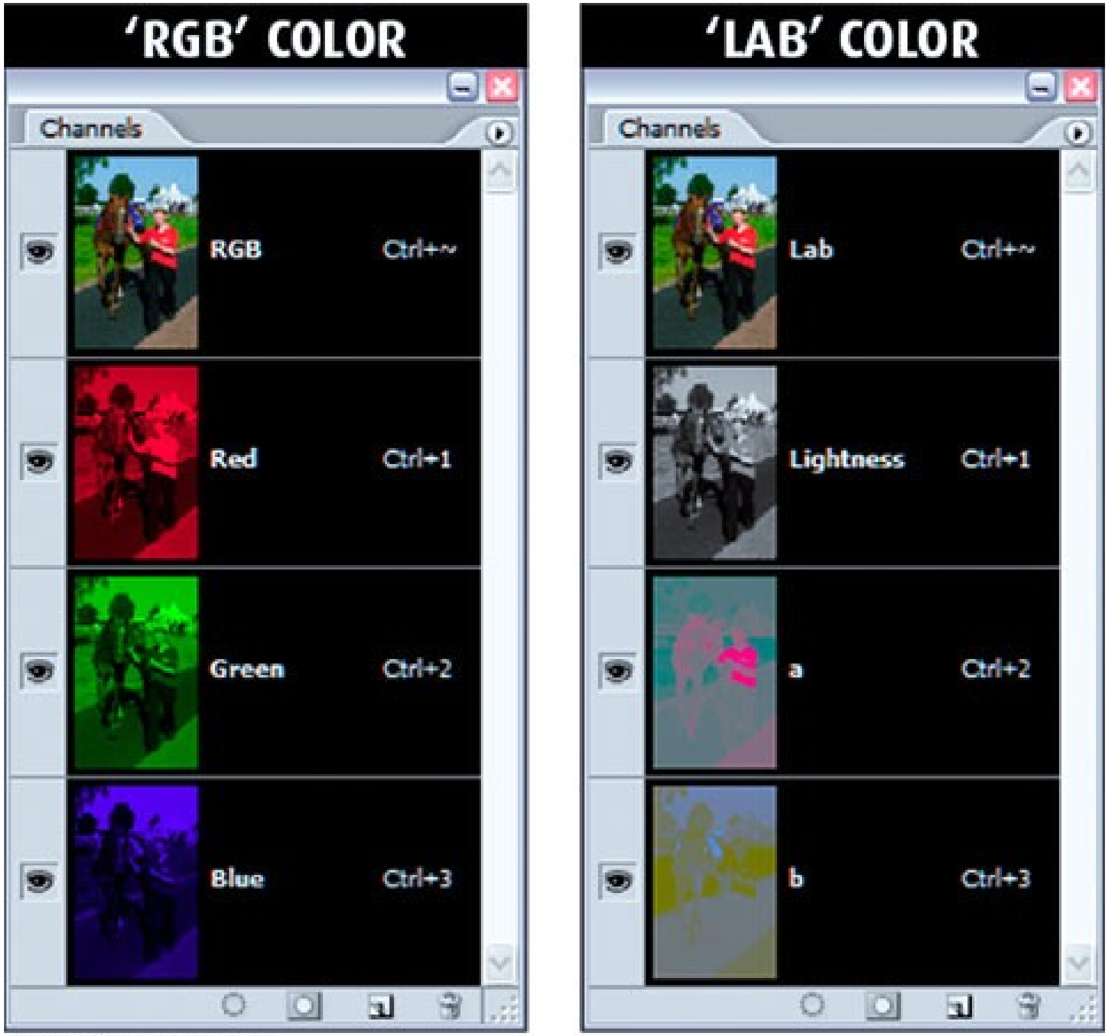
RGB → LAB

$$L^* = 116 \times (\frac{Y}{Yn})^{\frac{1}{3}} - 16 \text{ voor } \frac{Y}{Yn} > 0,008856$$

$$L^* = 903,3 \times \frac{Y}{Yn}$$

$$a^* = 500 \times (f(\frac{X}{Xn}) - f(\frac{Y}{Yn}))$$

$$b^* = 200 \times (f(\frac{Y}{Yn}) - f(\frac{Z}{Zn}))$$



Helps also with deep learning

Table 1. Comparison of results for different color spaces on CIFAR-10 with simple CNN

Color Space	Accuracy	Time
RGB	78.89	26 secs
HSV	78.57	26 secs
YUV	78.89	26 secs
LAB	**80.43**	26 secs
YIQ	78.79	26 secs
XYZ	78.72	26 secs
YPbPr	78.78	26 secs
YCbCr	78.81	26 secs
HED	78.98	26 secs
LCH	78.82	26 secs

https://arxiv.org/pdf/1902.00267.pdf

Learning

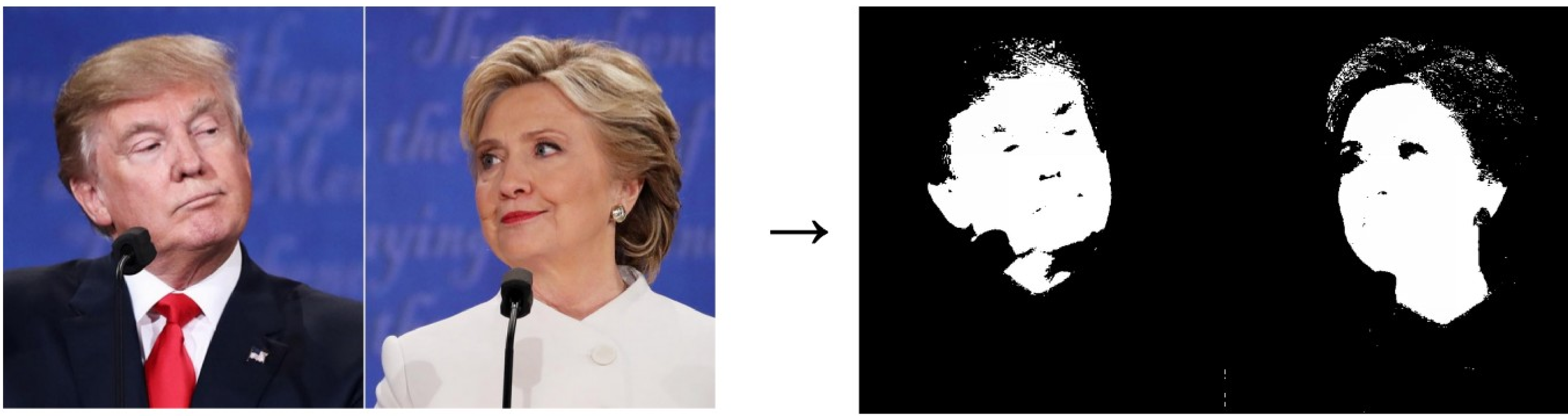
Carefully designed...

$$
\begin{aligned}
L^* &= 116 \times \left(\frac{Y}{Yn}\right)^{\frac{1}{3}} - 16 \text{ voor } \frac{Y}{Yn} > 0,008856 \\
L^* &= 903,3 \times \frac{Y}{Yn} \\
a^* &= 500 \times \left(f\left(\frac{X}{Xn}\right) - f\left(\frac{Y}{Yn}\right)\right) \\
b^* &= 200 \times \left(f\left(\frac{Y}{Yn}\right) - f\left(\frac{Z}{Zn}\right)\right)
\end{aligned}
$$

But can also be learned by deep learning...

How? 1X1 convolutions

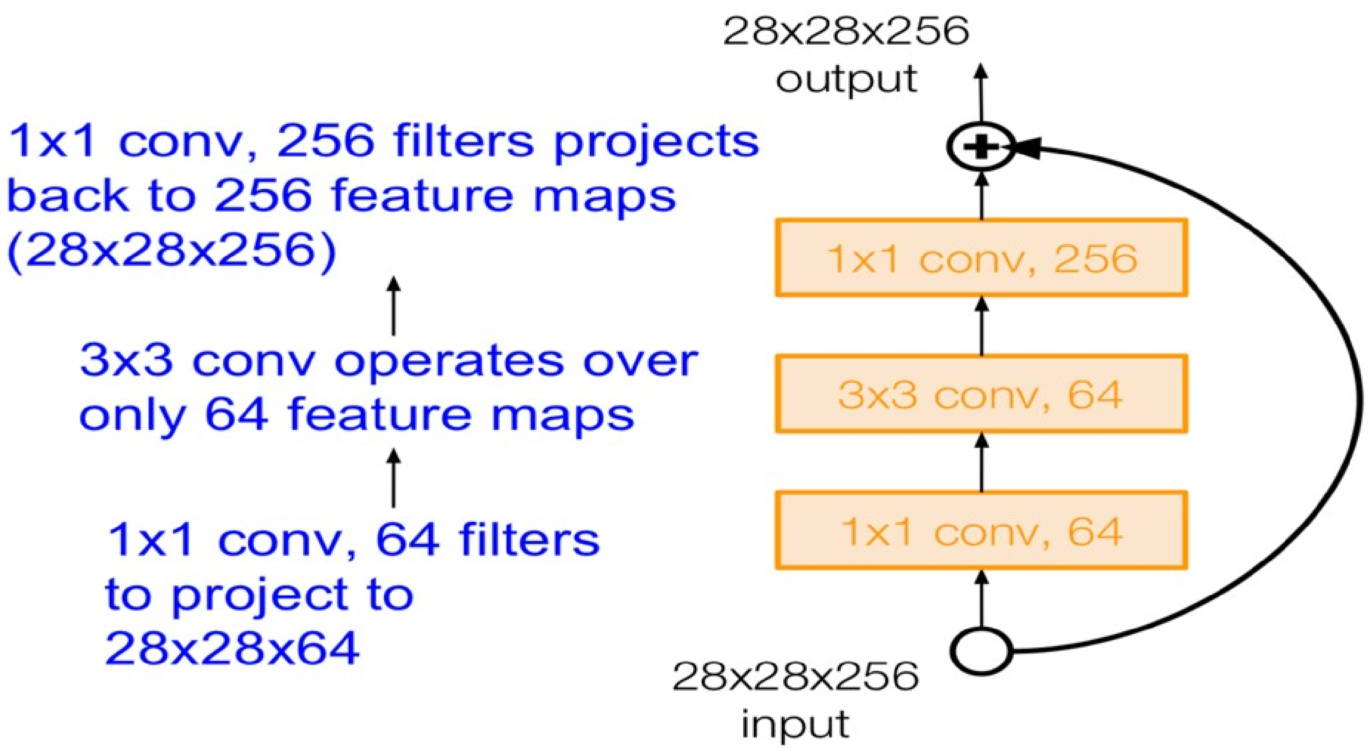
Example of learned colorspace



Other use for 1x1 convolution

Data compression

ResNet

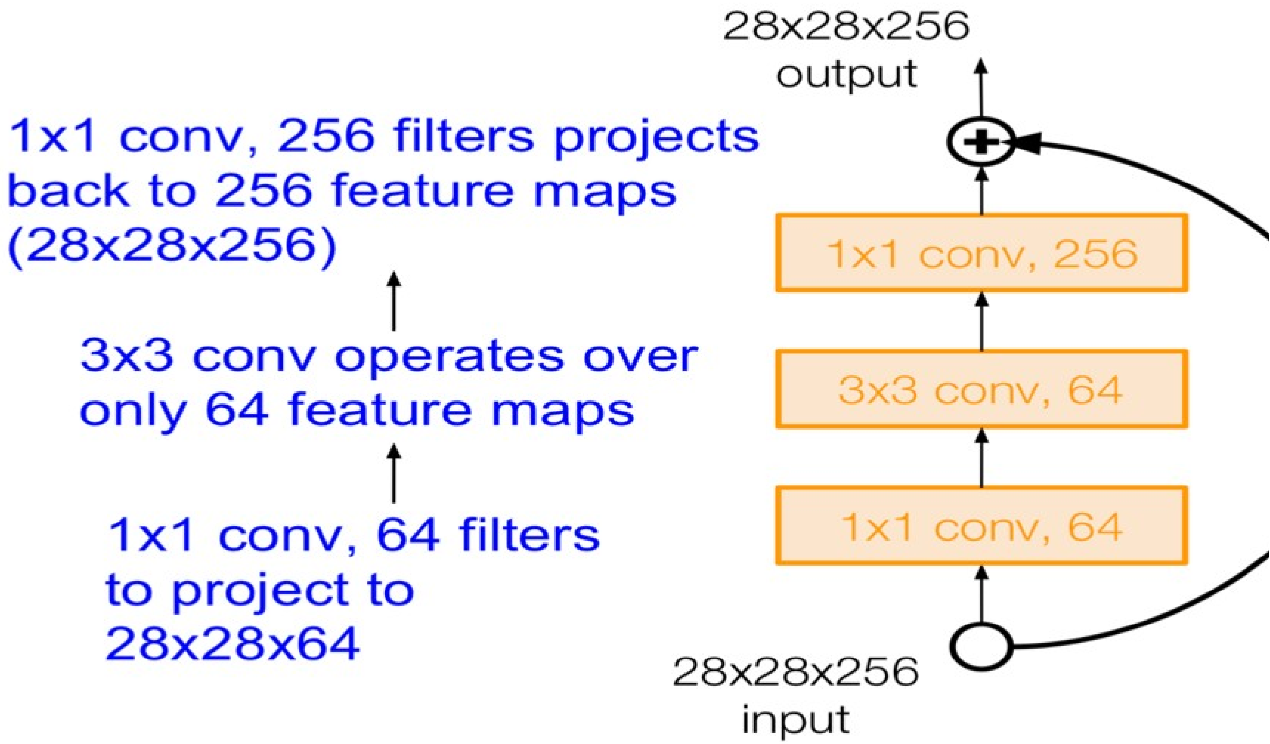


https://medium.com/analytics-vidhya/talented-mr-1x1-comprehensive-look-at-1x1-convolution-in-deep-learning-f6b355825578

Dimension reduction

3x3 calculations are expensive

Keeps height and width of image but reduce nmbr of features



SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE

Forrest N. Iandola[1], **Song Han**[2], **Matthew W. Moskewicz**[1], **Khalid Ashraf**[1],
William J. Dally[2], **Kurt Keutzer**[1]
[1]DeepScale* & UC Berkeley [2]Stanford University
{forresti, moskewcz, kashraf, keutzer}@eecs.berkeley.edu
{songhan, dally}@stanford.edu

ABSTRACT

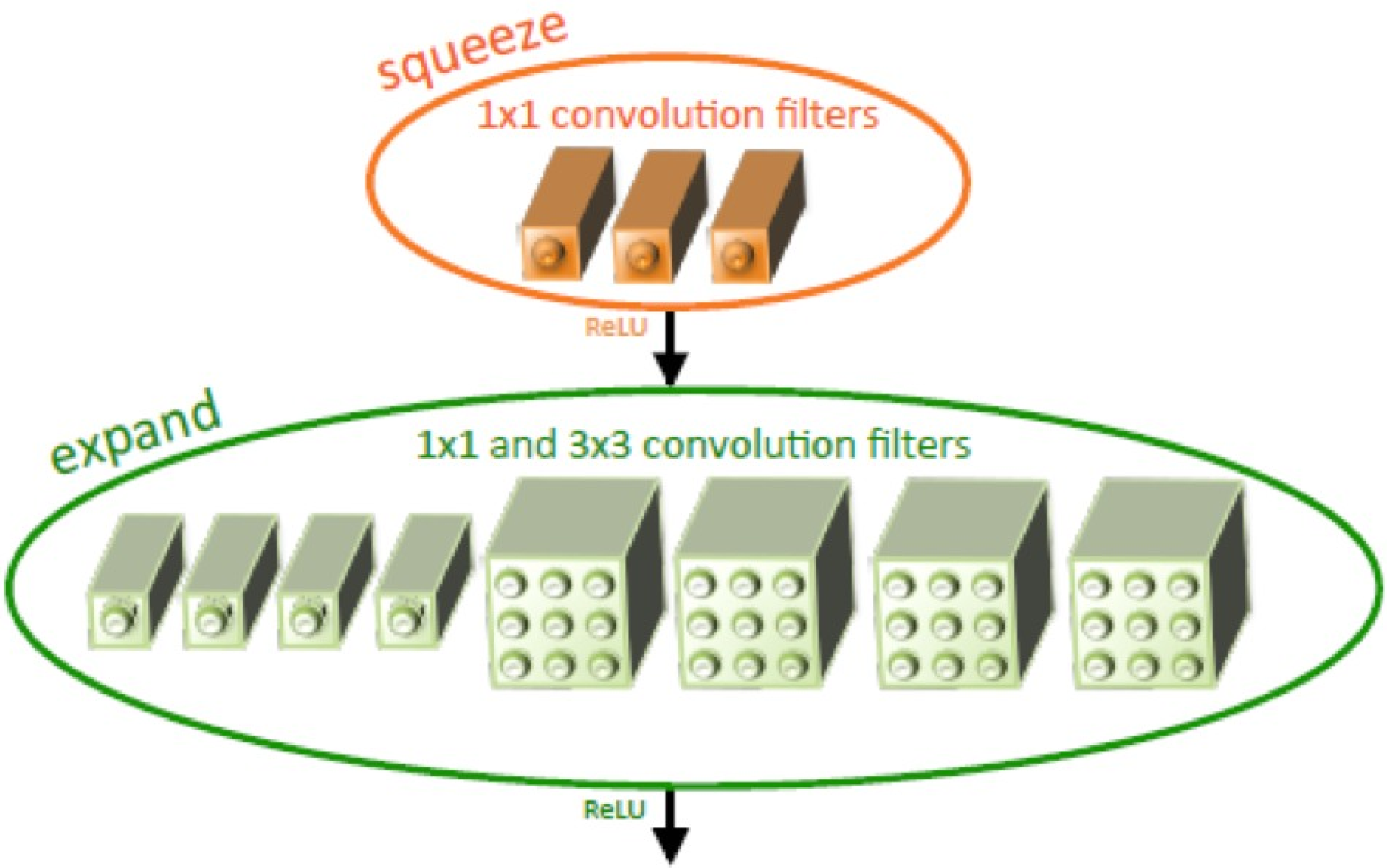
Recent research on deep convolutional neural networks (CNNs) has focused primarily on improving accuracy. For a given accuracy level, it is typically possible to identify multiple CNN architectures that achieve that accuracy level. With equivalent accuracy, smaller CNN architectures offer at least three advantages: (1) Smaller CNNs require less communication across servers during distributed training. (2) Smaller CNNs require less bandwidth to export a new model from the cloud to an autonomous car. (3) Smaller CNNs are more feasible to deploy on FPGAs and other hardware with limited memory. To provide all of these advantages, we propose a small CNN architecture called SqueezeNet. SqueezeNet achieves AlexNet-level accuracy on ImageNet with 50x fewer parameters. Additionally, with model compression techniques, we are able to compress SqueezeNet to less than 0.5MB (510× smaller than AlexNet).

The SqueezeNet architecture is available for download here: https://github.com/DeepScale/SqueezeNet

1 INTRODUCTION AND MOTIVATION

Much of the recent research on deep convolutional neural networks (CNNs) has focused on increasing accuracy on computer vision datasets. For a given accuracy level, there typically exist multiple CNN architectures that achieve that accuracy level. Given equivalent accuracy, a CNN architecture

Paper



https://openreview.net/pdf?id=S1xh5sYgx

Results

Result:

<u>50 times ! less parameters</u> than AlexNet with same <u>accuracy</u>!!!

Last layers

Convolutional neural network:

Part1:	Multiple convolution layers with pooling and activation functions.
Part 2:	Followed by 1 or more fully connected layers

Last layers

Convolutional neural network:

Part1: Multiple convolution layers with pooling and activation functions.

Part 2: Followed by 1 or more fully connected layers

Downside: Input dimensions (input width and height) have to be fixed because of the linear layers...

Solved by using a conv layer with kernel size equal to the size of the image.

Last layers

Convolutional neural network:

Part1: Multiple convolution layers with pooling and activation functions.

~~Part 2:~~ ~~Followed by flattening and multiple fully connected layers~~

Part 2: Use 1x1 convolution instead!

(first layer should have kernel size same as width and height last convolutional layer)

Last layers

This step is important. It changes a 10 x 10 image into a 1 x 1 image with many feature channels. Please write it as step 2 and 1x1 conv as step 3. Otherwise, students may think 1x1 conv can change image width-height. We need to be clear that 1x1 conv can only reduce or increase or recombine the original feature space.

Convolutional neural network:

Part1: Multiple convolution layers with pooling and activation functions.

~~Part 2: Followed by flattening and multiple fully connected layers~~

Part 2: Use 1x1 convolution instead!

“first layer” is confusing :)

(first layer should have kernel size same as width and height last convolutional layer)

Conv2D(3,16,3) → Conv2D(3,16,3) → nn.Flatten()

→ nn.Linear(~~100~~,64) → nn.Linear(64,32) → nn.Linear(32,10)

1600?

or

Conv2D(3,16,3) → Conv2D(3,16,3) → nn.Conv2d(16,64, 10)

→ nn.Conv2d(64,32, ksize=1) → nn.Conv2d(32,10, ksize=1)

Other uses

?

Glow: Generative Flow with Invertible 1×1 Convolutions

Diederik P. Kingma[*], Prafulla Dhariwal[*]
OpenAI, San Francisco

Abstract

Flow-based generative models (Dinh et al., 2014) are conceptually attractive due to tractability of the exact log-likelihood, tractability of exact latent-variable inference, and parallelizability of both training and synthesis. In this paper we propose *Glow*, a simple type of generative flow using an invertible 1×1 convolution. Using our method we demonstrate a significant improvement in log-likelihood on standard benchmarks. Perhaps most strikingly, we demonstrate that a generative model optimized towards the plain log-likelihood objective is capable of efficient realistic-looking synthesis and manipulation of large images. The code for our model is available at `https://github.com/openai/glow`.

1 Introduction

Two major unsolved problems in the field of machine learning are (1) data-efficiency: the ability to learn from few datapoints, like humans; and (2) generalization: robustness to changes of the task or its context. AI systems, for example, often do not work at all when given inputs that are different from their training distribution. A promise of *generative models*, a major branch of machine learning,

[*]Equal contribution.

Preprint. Work in progress.